Optimizing DMA Data Transfers for Embedded Multi-Cores

Selma Saïdi

Jury members:

Oded Maler: Dir. de these
Luca Benini: Rapporteur
Eric Flamand: Examinateur
Ahmed Bouajjani: President du Jury
Albert Cohen: Rapporteur
Bruno Jego: Examinateur

Context of the Thesis

- Ph.D CIFRE with STMicroelectronics, supervised by,
  - Oded Maler, Verimag,
  - Bruno Jego and in collaboration with Thierry Lepley, STMicroelectronics
- Minalogic project ATHOLE
  - low-power multi-core platform for embedded systems
  - partners: ST, CEA, Thales, CWS, Verimag

Outline

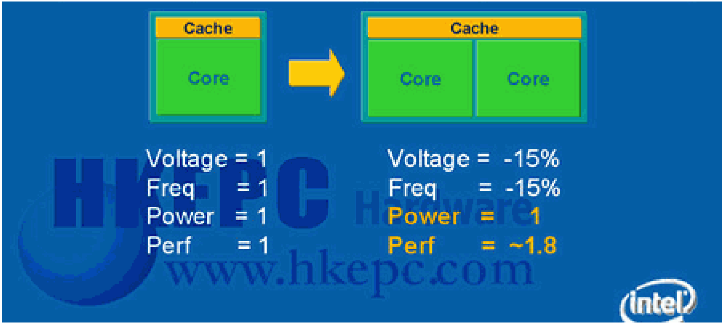
Embedded Systems

There is an increasing requirement for performance under low power constraints:

- Need to integrate more functionnalities in Embedded devices,
- Applications are becoming more computationaly intensive and power hungry,

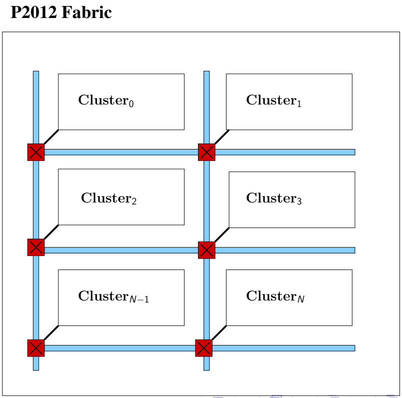
The Emergence of Multicore Architectures

- Running 2 processors in the same chip at half the speed will be less energy consuming and equally performant,



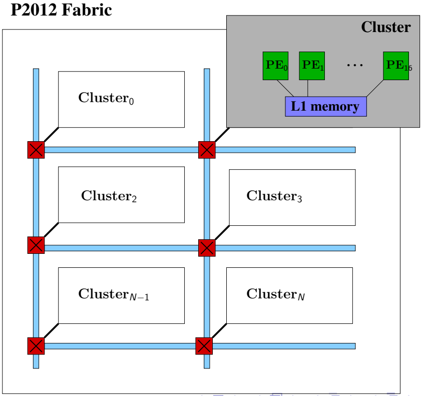
Embedded Multicore Architectures:

- Platform 2012: a manycore computation fabric,



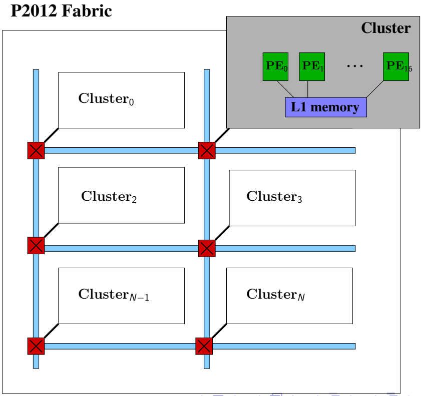
Embedded Multicore Architectures:

- Platform 2012: a manycore computation fabric,



Embedded Multicore Architectures:

- Platform 2012: a manycore computation fabric,
- main characteristic: explicitly managed Memories:

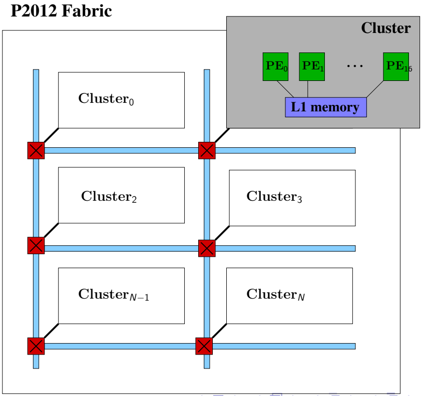


Embedded Multicore Architectures:

- Platform 2012: a manycore computation fabric,
- main characteristic: explicitly managed Memories:

Features:

1. Scratchpad memories (No caches),

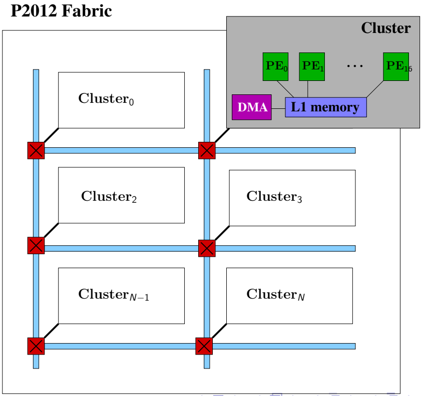


Embedded Multicore Architectures:

- Platform 2012: a manycore computation fabric,
- main characteristic: explicitly managed Memories:

Features:

1. Scratchpad memories (No caches)
2. DMA engine: a hardware for managing data transfers,

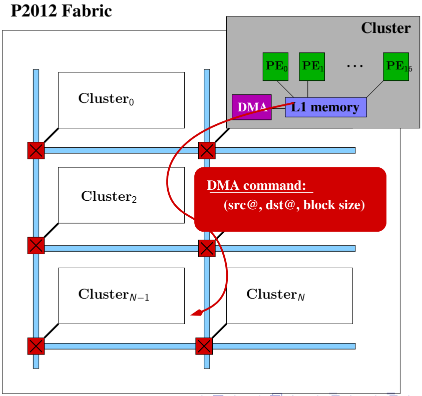


Embedded Multicore Architectures:

- Platform 2012: a manycore computation fabric,
- main characteristic: explicitly managed Memories:

Features:

1. Scratchpad memories (No caches)
2. DMA engine: a hardware for managing data transfers,

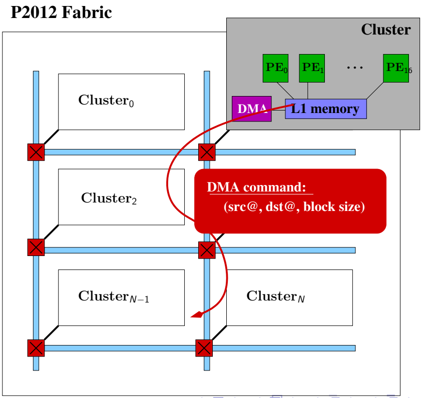


Embedded Multicore Architectures:

- Platform 2012: a manycore computation fabric,
- main characteristic: explicitly managed Memories:

Features:

1. Scratchpad memories (No caches)
2. DMA engine: a hardware for managing data transfers,
3. data transfers are explicitly managed by the software,

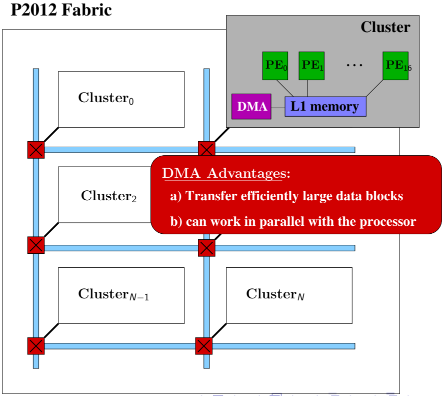


Embedded Multicore Architectures:

- Platform 2012: a manycore computation fabric,
- main characteristic: explicitly managed Memories:

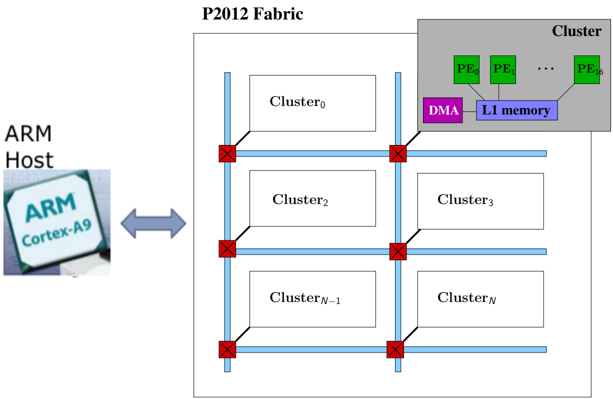
Features:

1. Scratchpad memories (No caches)
2. DMA engine: a hardware for managing data transfers,
3. data transfers are explicitly managed by the software,



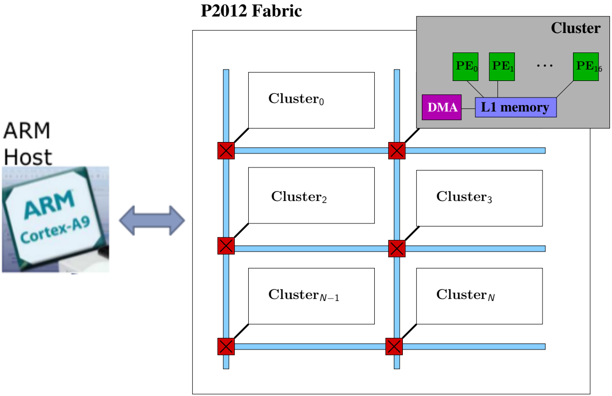
Embedded Multicore Architectures:

- acts as a general purpose programmable accelerator:
  ⇒ Heterogeneous Multicore Architectures,



Embedded Multicore Architectures:

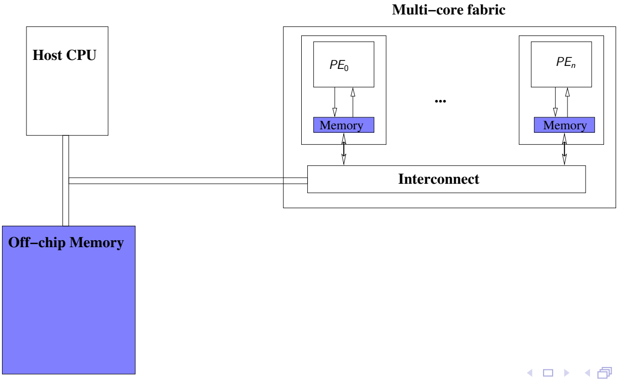
- acts as a general purpose programmable accelerator:
  ⇒ Heterogeneous Multicore Architectures,



This is the class of architectures in which we are interested !!

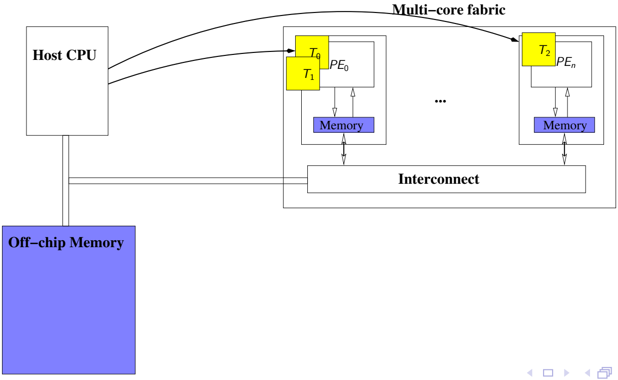
Heterogeneous Multi-core Architectures

- a powerful host processor and a multi-core fabric to accelerate computationally heavy kernels.



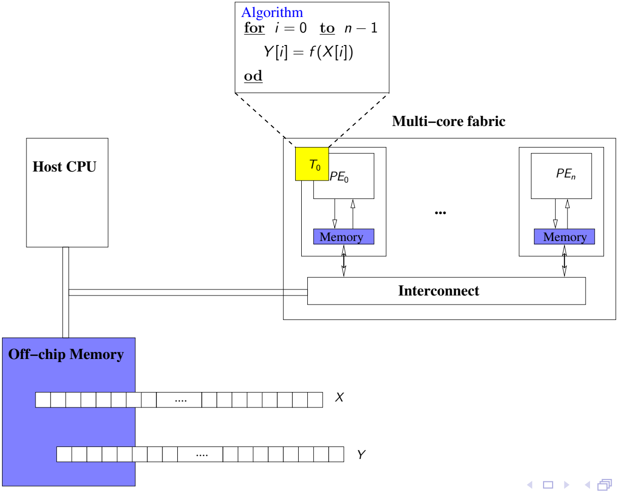
Heterogeneous Multi-core Architectures

- a powerful host processor and a multi-core fabric to accelerate computationally heavy kernels.



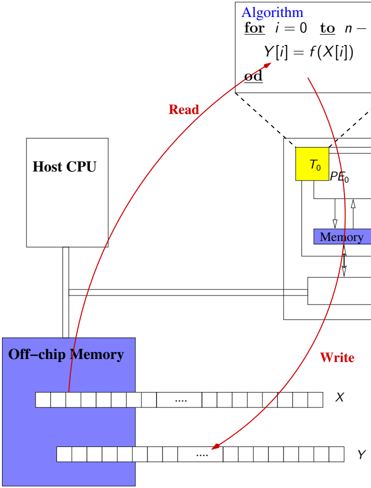
Heterogeneous Multi-core Architectures

- Offloadable kernels work on large data sets, initially stored in a distant off-chip memory.



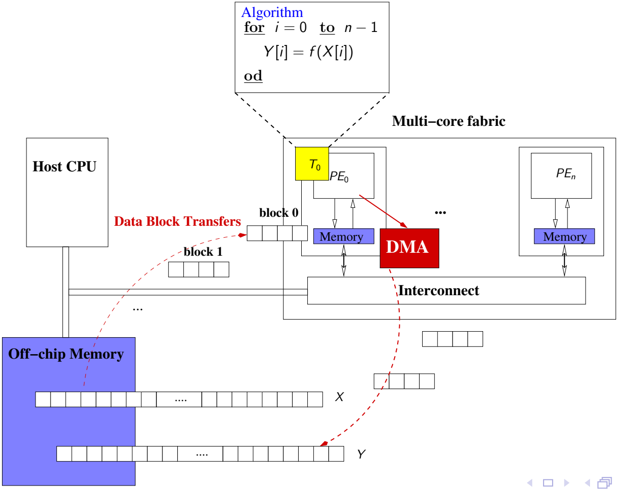
Heterogeneous Multi-core Architectures

- High off-chip memory latency: accessing off-chip data is very costly



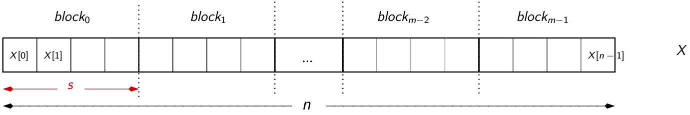
Heterogeneous Multi-core Architectures

- Data is transferred to a closer but smaller on-chip memory, using DMAs (Direct Memory Access).



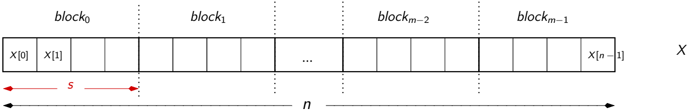
DMA Data Transfers: Single Buffering

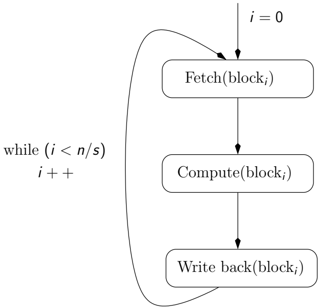
s: number of array elements in one block,



DMA Data Transfers: Single Buffering

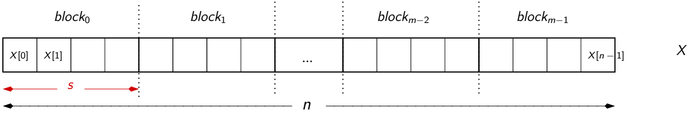
s: number of array elements in one block,

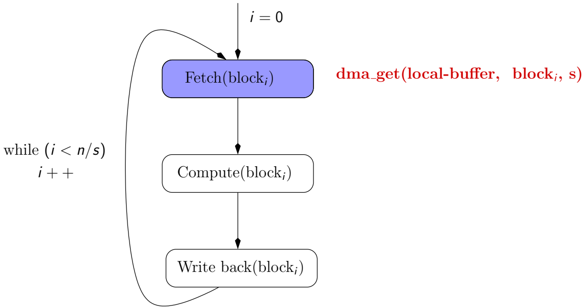




DMA Data Transfers: Single Buffering

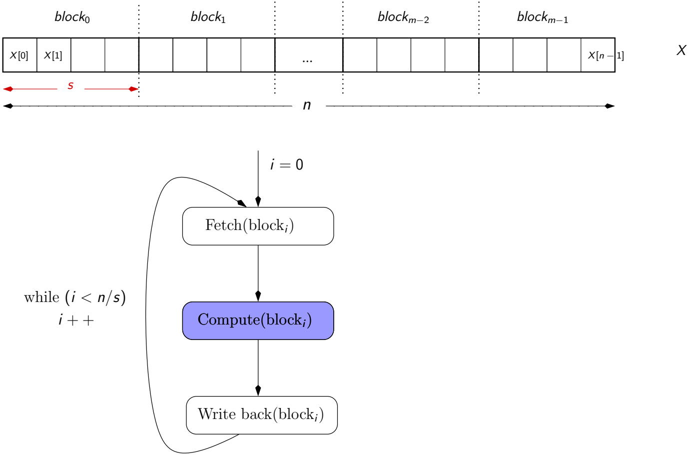
s: number of array elements in one block,





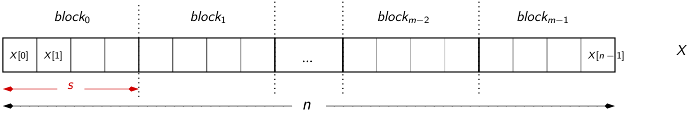
DMA Data Transfers: Single Buffering

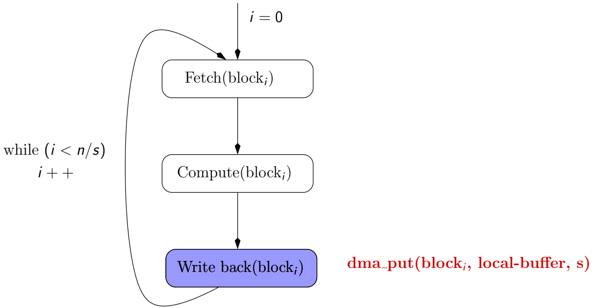
s: number of array elements in one block,



DMA Data Transfers: Single Buffering

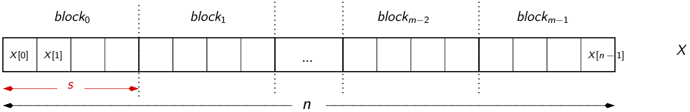
s: number of array elements in one block,

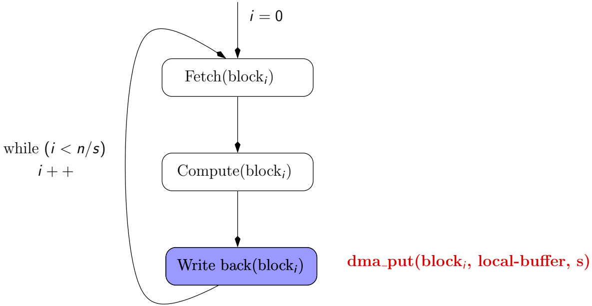




DMA Data Transfers: Single Buffering

s: number of array elements in one block,

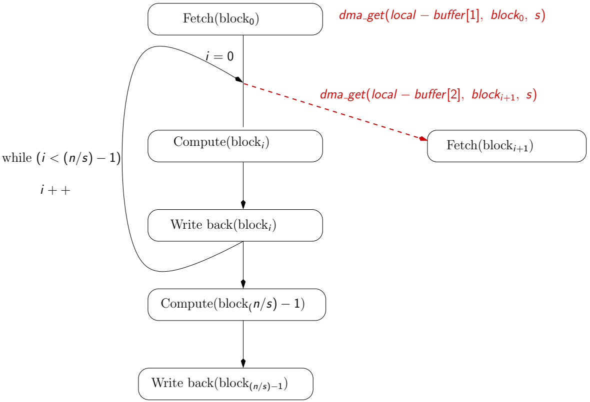




- Sequential execution of computations and data transfers.

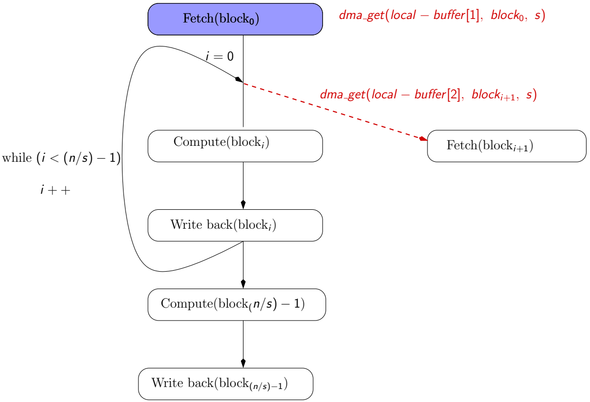
DMA Data Transfers: Double Buffering

Asynchronous DMA calls:



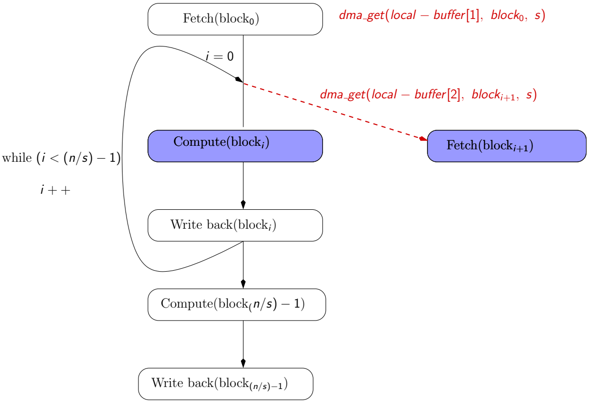
DMA Data Transfers: Double Buffering

Asynchronous DMA calls:



DMA Data Transfers: Double Buffering

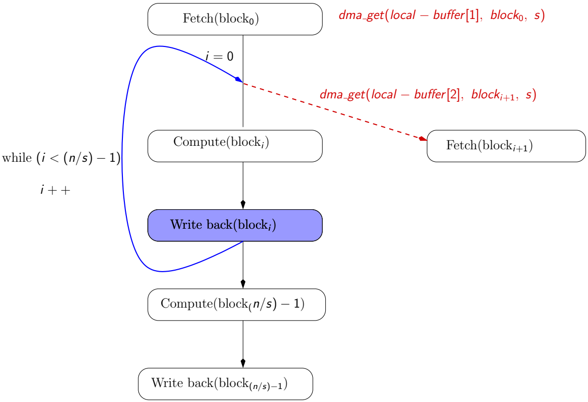
Asynchronous DMA calls:



- Overlap of computations and data transfers.

DMA Data Transfers: Double Buffering

Asynchronous DMA calls:

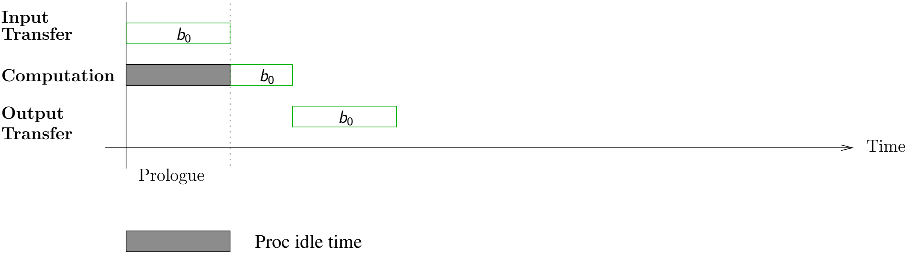


- Overlap of computations and data transfers.

Double Buffering Pipelined Execution

Overlap of,

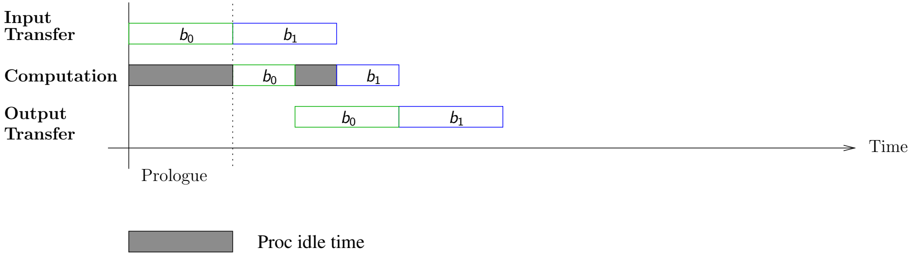
- *Computation* of current block,
- *Transfer* of next block.



Double Buffering Pipelined Execution

Overlap of,

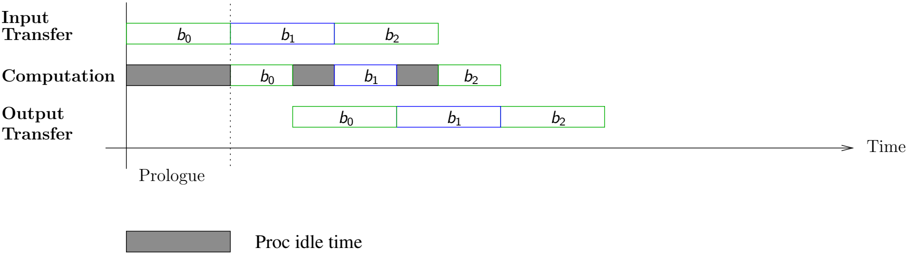
- *Computation* of current block,
- *Transfer* of next block.



Double Buffering Pipelined Execution

Overlap of,

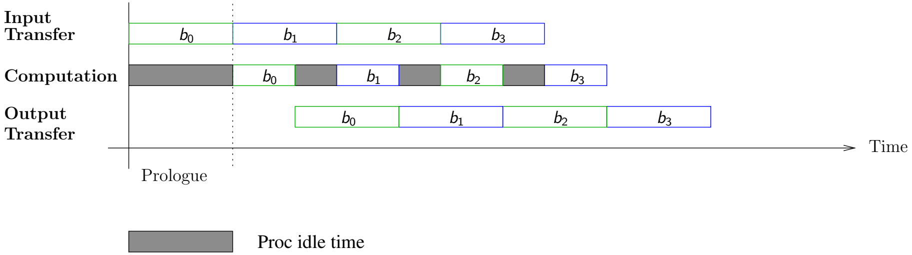
- *Computation* of current block,
- *Transfer* of next block.



Double Buffering Pipelined Execution

Overlap of,

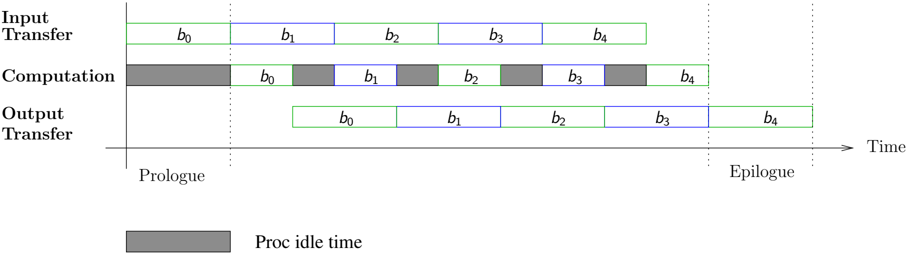
- *Computation* of current block,
- *Transfer* of next block.



Double Buffering Pipelined Execution

Overlap of,

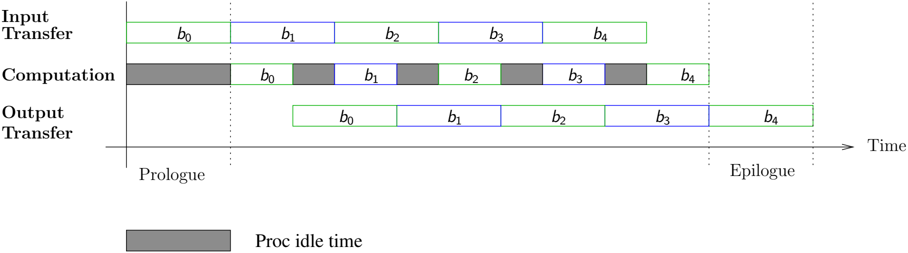
- *Computation* of current block,
- *Transfer* of next block.



Double Buffering Pipelined Execution

Overlap of,

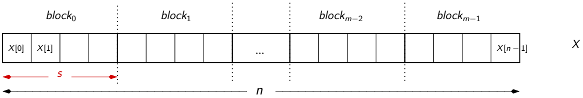
- *Computation* of current block,
- *Transfer* of next block.



Performance can be further improved by an appropriate choice of data granularity.

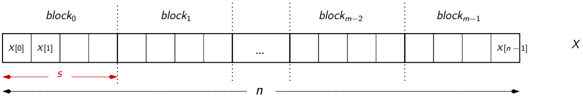
Granularity of Transfers

- 1Dim Data:
  block size s

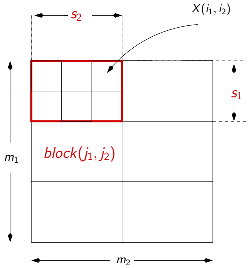


Granularity of Transfers

- 1Dim Data:
  block size s

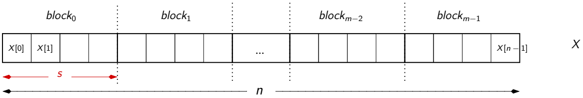


- 2Dim Data:
  block shape
  (s_1, s_2)

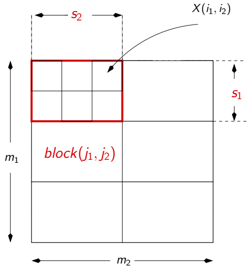


Granularity of Transfers

- 1Dim Data:
  block size s



- 2Dim Data:
  block shape
  (s_1, s_2)



Contribution:

We derive optimal granularity for 1D and 2D DMA transfers,

Our Contribution:

We derive optimal granularity for 1D and 2D DMA transfers,

- 1Dim data work was published in Hipeac 2012,
  S.Saidi, P.tendulkar, T.Lepley, O.Maler, "Optimizing explicit data transfers for data parallel applicationson the Cell architecture "
- 2D data work was published in DSD 2012,
  S.Saidi, P.tendulkar, T.Lepley, O.Maler, "Optimal 2D Data Partitioning for DMA Transfers on MPSoCs"
  - extended version of the paper submitted to: "Embedded Hardware Design: Microprocessors and Microsystems" Journal.

Outline

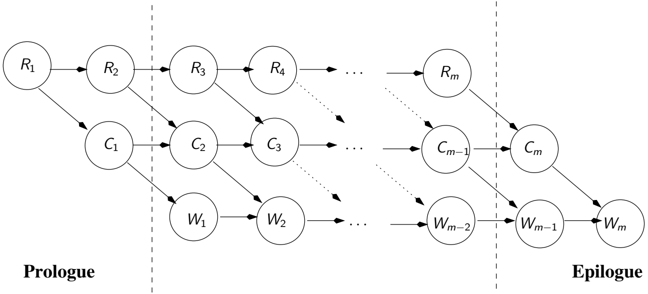
1. Context and Motivation
2. Contribution
   - Problem Definition
   - Optimal Granularity for a Single Processor
     - 1Dim Data
     - 2Dim Data
   - Multiple Processors
   - Shared Data
3. Experiments on the Cell.BE
4. The move towards Platform 2012
5. Conclusions and Perpectives

Outline

Outline

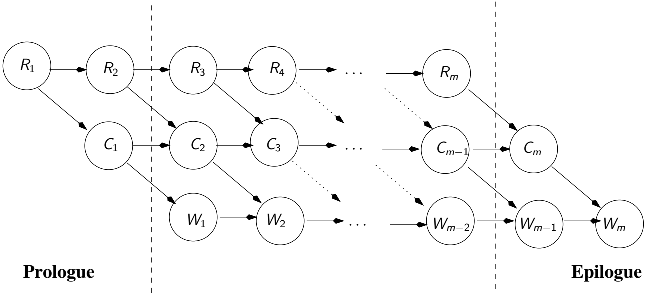
Software Pipelining

- We want to optimize execution of the pipeline.



Software Pipelining

- We want to optimize execution of the pipeline.



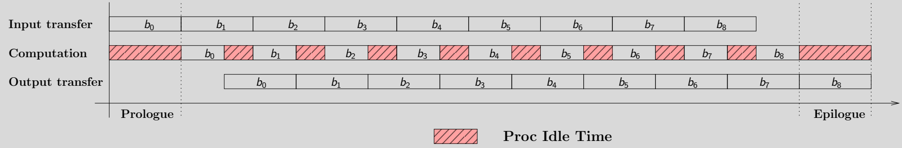
Optimal Granularity:

What is the Granularity choice that optimizes performance?

Computation Regime and Transfer Regime

- T and C: Transfer and Computation time of a block

Transfer Regime $T > C$:

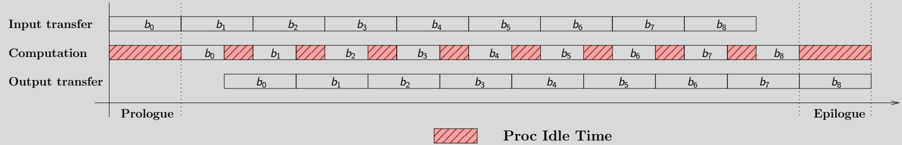


Computation Regime $C > T$:

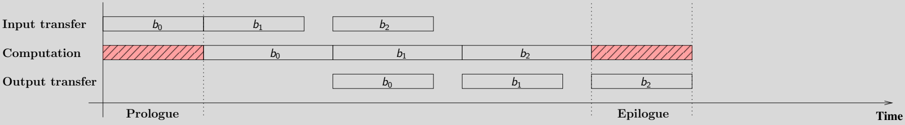
Computation Regime and Transfer Regime

- T and C: Transfer and Computation time of a block

Transfer Regime $T > C$:

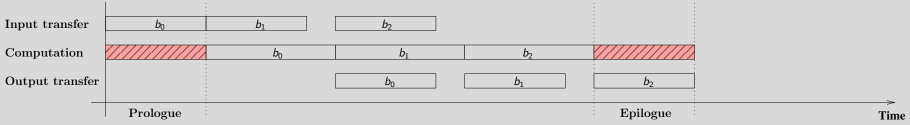


Computation Regime $C > T$:

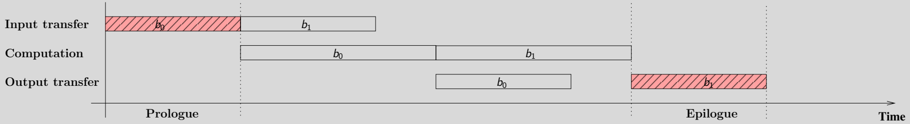


In the Computation Regime:

Granularity s:



Granularity s': $s' > s$:



Optimal Granularity: Problem Formulation

1Dim Data: block size

Find s^* such that,

$$Min\ T(s) \quad \text{s.t.}$$

$$T(s) \leq C(s)$$
$$(s) \in [1..n]$$
$$s \leq M$$

2Dim Data: block shape

Optimal Granularity: Problem Formulation

1Dim Data: block size

Find s^* such that,

$$Min\ \ T(s) \ \ \text{s.t.}$$

$$T(s) \leq C(s)$$
$$(s) \in [1..n]$$
$$s \leq M$$

2Dim Data: block shape

Find (s_1^*, s_2^*) such that,

$$Min\ \ T(s_1, s_2) \ \ \text{s.t.}$$

$$T(s_1, s_2) \leq C(s_1, s_2)$$
$$(s_1, s_2) \in [1..n_1] \times [1..n_2]$$
$$s_1 \times s_2 \leq M$$

Outline

Outline

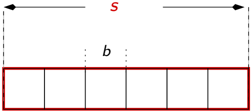
Optimal Granularity : 1Dim Data

Characterization of Computation and Transfer Time:

Optimal Granularity : 1Dim Data

Characterization of Computation and Transfer Time:

- s: nb array elements clustered in one (Contiguous) block,



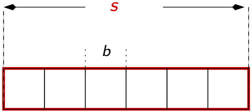
Computation time C(s):

DMA Transfer time T(s):

Optimal Granularity : 1Dim Data

Characterization of Computation and Transfer Time:

- s: nb array elements clustered in one (Contiguous) block,



Computation time C(s):

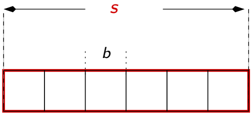
- ω: time to compute one element,

DMA Transfer time T(s):

Optimal Granularity : 1Dim Data

Characterization of Computation and Transfer Time:

- s: nb array elements clustered in one (Contiguous) block,



Computation time C(s):

- ω: time to compute one element,

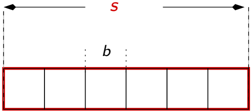
$$C(s) = \omega \cdot s$$

DMA Transfer time T(s):

Optimal Granularity : 1Dim Data

Characterization of Computation and Transfer Time:

- s: nb array elements clustered in one (Contiguous) block,



Computation time C(s):

- ω: time to compute one element,

$$C(s) = \omega \cdot s$$

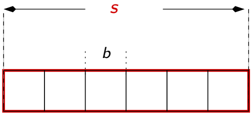
DMA Transfer time T(s):

- I: fixed DMA initialization cost,

Optimal Granularity : 1Dim Data

Characterization of Computation and Transfer Time:

- s: nb array elements clustered in one (Contiguous) block,



Computation time C(s):

- ω: time to compute one element,

$$C(s) = \omega \cdot s$$

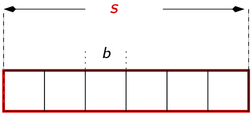
DMA Transfer time T(s):

- I: fixed DMA initialization cost,
- α: transfer cost per byte,

Optimal Granularity : 1Dim Data

Characterization of Computation and Transfer Time:

- s: nb array elements clustered in one (Contiguous) block,



Computation time C(s):

- ω: time to compute one element,

$$C(s) = \omega \cdot s$$

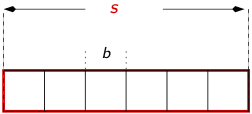
DMA Transfer time T(s):

- I: fixed DMA initialization cost,
- α: transfer cost per byte,
- b: size of one array element,

Optimal Granularity : 1Dim Data

Characterization of Computation and Transfer Time:

- s: nb array elements clustered in one (Contiguous) block,



Computation time C(s):

- ω: time to compute one element,

$$C(s) = \omega \cdot s$$

DMA Transfer time T(s):

- I: fixed DMA initialization cost,
- α: transfer cost per byte,
- b: size of one array element,

$$T(s) = I + \alpha \cdot b \cdot s$$

Optimal Granularity : 1Dim Data

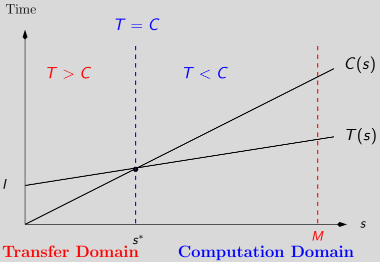
Pb Formulation

$Min\ \ T(s)$ s.t.

$T(s) \leq C(s)$
$s \in [1..n]$
$s \leq M$

- s: block size
- M: Memory limitation

Optimal Granularity s^*:



Optimal Granularity : 1Dim Data

Pb Formulation

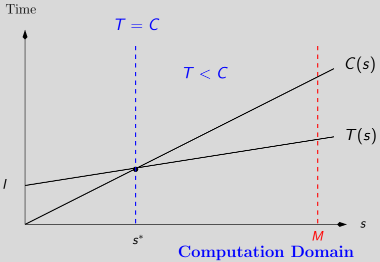
$Min\ T(s)$ s.t.

$T(s) \leq C(s)$
$s \in [1..n]$
$s \leq M$

- s: block size
- M: Memory limitation

Optimal Granularity s^*:

$$C(s^*) = T(s^*)$$

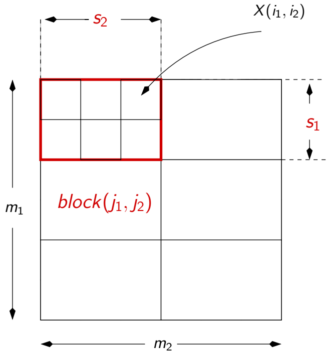


Outline

Optimal Granularity : 2Dim Data

Characterization of Computation and Transfer Time:

- $(s_1 \times s_2)$: nb array elements clustered in one (square) block,

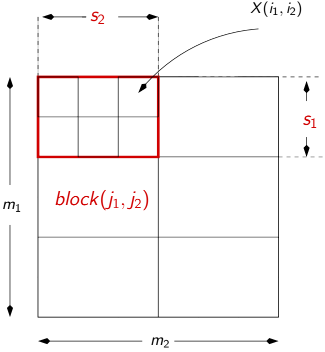


Computation time $C(s_1, s_2)$:

Optimal Granularity : 2Dim Data

Characterization of Computation and Transfer Time:

- $(s_1 \times s_2)$: nb array elements clustered in one (square) block,



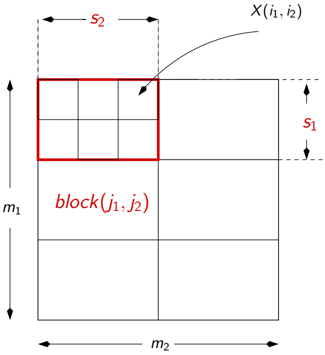
Computation time $C(s_1, s_2)$:

- ω: time to compute one element,

Optimal Granularity : 2Dim Data

Characterization of Computation and Transfer Time:

- $(s_1 \times s_2)$: nb array elements clustered in one (square) block,

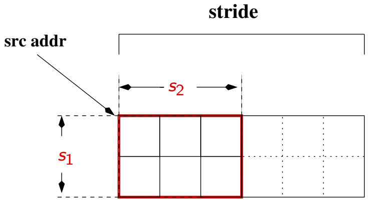


Computation time $C(s_1, s_2)$:

- ω: time to compute one element,

$$C(s_1, s_2) = \omega \cdot s_1 \cdot s_2$$

Optimal Granularity : 2Dim Data

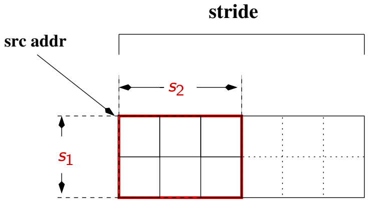


Strided DMA Transfer time $T(s_1, s_2)$:

- l_1: transfer cost overhead per line,

$$T(s_1, s_2) = I + l_1 s_1 + \alpha \cdot b \cdot s_1 \cdot s_2$$

Optimal Granularity : 2Dim Data



Strided DMA Transfer time $T(s_1, s_2)$:

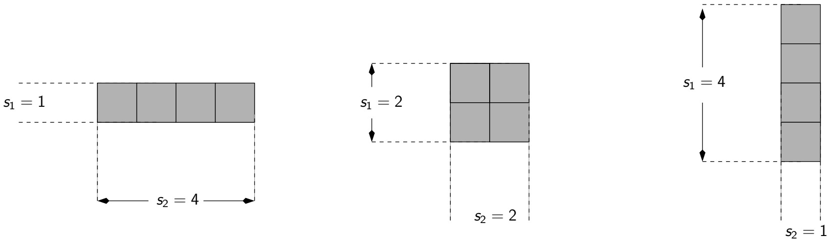
- l_1: transfer cost overhead per line,

$$T(s_1, s_2) = I + l_1 s_1 + \alpha \cdot b \cdot s_1 \cdot s_2$$

Strided DMA transfers are costlier than contiguous transfers

Influence of the Block Shape on the DMA Transfer Cost

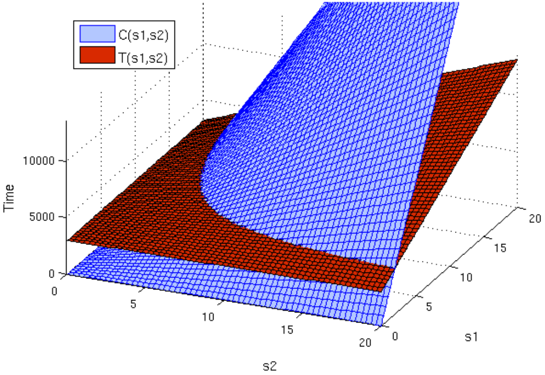
- Different block shapes with same area **BUT** different DMA transfer time,



$(s_1, s_2) = (1, 4)$ $(s_1, s_2) = (2, 2)$ $(s_1, s_2) = (4, 1)$

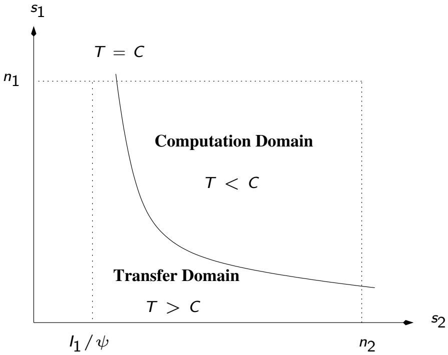
Optimal Granularity : 2Dim Data

- $C(s_1, s_2)$: computation time of a block,
- $T(s_1, s_2)$: transfer time of a block,



Optimal Granularity : 2Dim Data

- $C(s_1, s_2)$: computation time of a block,
- $T(s_1, s_2)$: transfer time of a block,



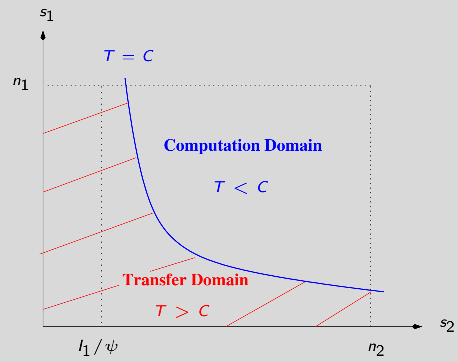
Optimal Granularity : 2Dim Data

Pb Formulation

$Min\ T(s_1, s_2)$ s.t.

$T(s_1, s_2) \leq C(s_1, s_2)$
$(s_1, s_2) \in [1..n_1] \times [1..n_2]$
$s_1 \times s_2 \leq M$

- s_1: block height
- s_2: block width



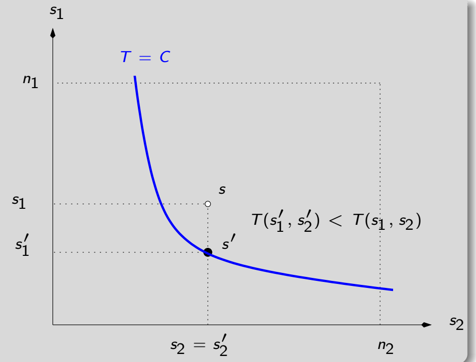
Optimal Granularity : 2Dim Data

Pb Formulation

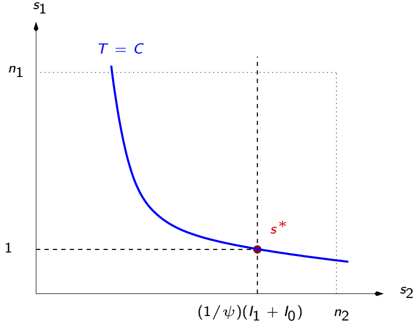
$Min\ T(s_1, s_2)$ s.t.

$T(s_1, s_2) \leq C(s_1, s_2)$
$(s_1, s_2) \in [1..n_1] \times [1..n_2]$
$s_1 \times s_2 \leq M$

- s_1: block height
- s_2: block width



Optimal Granularity : 2Dim Data



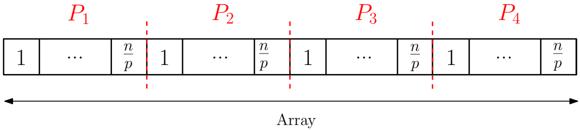
$$\begin{cases} \text{Block Height} : s_1^* = 1 \\ \text{Block Width} : s_2^* = (1/\psi)(l_1 + l_0) \end{cases}$$

Optimal granularity is the Contiguous block to reach the computation regime:

Outline

Multiple Processors

- Partitioning: p contiguous chunks of data



- Processors are identical: same local store capacity, same double buffering granularity...etc.

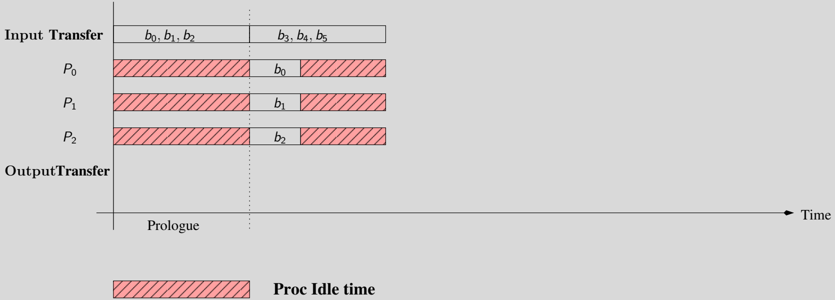
Multiple Processors

Pipelined execution for several processors:

- processors DMA requests are done concurrently,

Multiple Processors

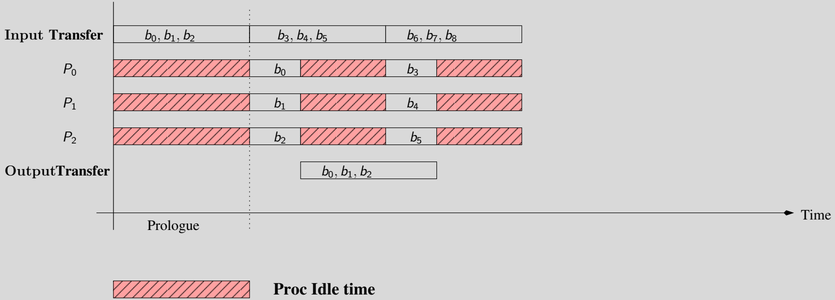
Pipelined execution for several processors:



- processors DMA requests are done concurrently,

Multiple Processors

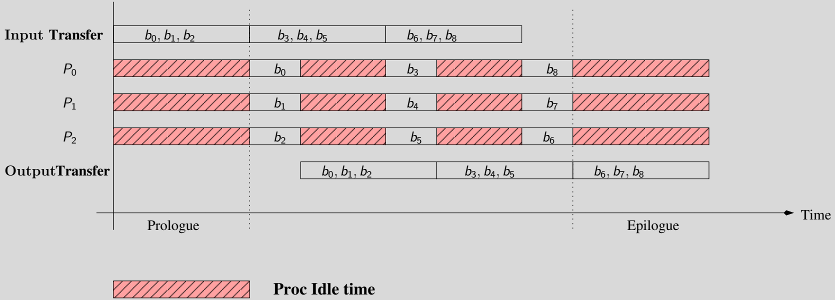
Pipelined execution for several processors:



- processors DMA requests are done concurrently,

Multiple Processors

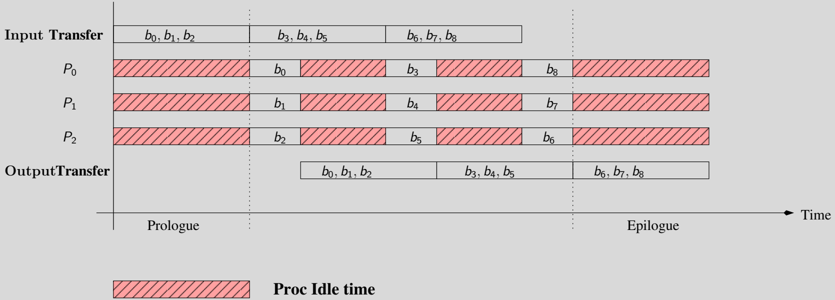
Pipelined execution for several processors:



- processors DMA requests are done concurrently,

Multiple Processors

Pipelined execution for several processors:



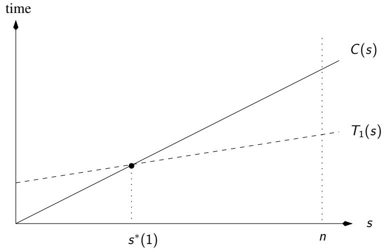
- processors DMA requests are done concurrently,

$$T(s, p) = I + \alpha(p) \cdot b \cdot s$$

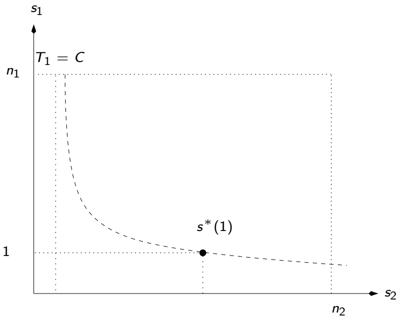
$\alpha(p)$: transfer cost per byte given contentions of p concurrent transfer requests.

Multiple Processors: Optimal Granularity

- Optimal granularity given p processors: $s^*(p)$,



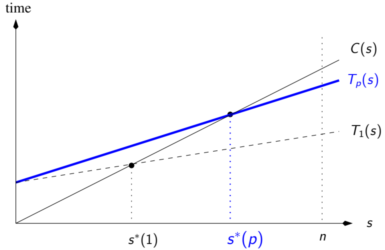
(a) One-dimensional data



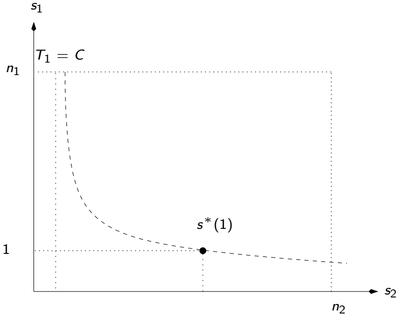
(b) Two-dimensional data

Multiple Processors: Optimal Granularity

- Optimal granularity given p processors: $s^*(p)$,



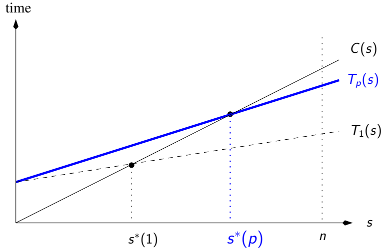
(a) One-dimensional data



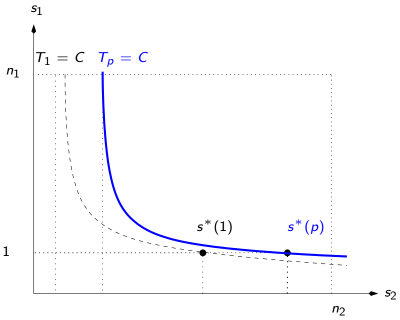
(b) Two-dimensional data

Multiple Processors: Optimal Granularity

- Optimal granularity given p processors: $s^*(p)$,



(a) One-dimensional data



(b) Two-dimensional data

Optimal Granularity increases with number of processors

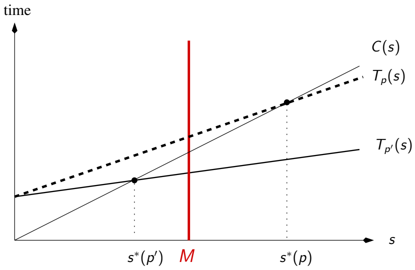
Summary:

We derived optimal granularity,

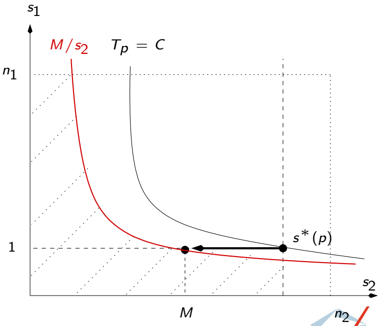
- main idea: balance between computation and transfer time of a block,
- 2D data: block shape influences transfer time (overhead per line, I_1)
- multiple processors: number of processors influence transfer time (with $\alpha(p)$)

Local Memory Constraint

- What if Optimal Granularity does not fit in Local memory?
  1. take available memory space,
  2. reduce the number of processors,



(a) One-dimensional data



(b) Two-dimensional data

Outline

Applications with shared data: 1Dim

- Data parallel loop with shared input data:

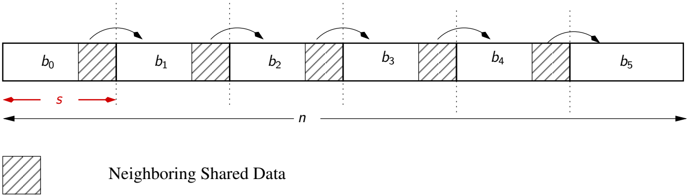
for $i := 0$ to $n-1$ do
  $Y[i] := f(X[i], V[i]);$ $V[i] = \{X[i-1], X[i-2], ..., X[i-k]\}$
od

Applications with shared data: 1Dim

- Data parallel loop with shared input data:

for $i := 0$ to $n-1$ do
$Y[i] := f(X[i], V[i]);$ $\quad V[i] = \{X[i-1], X[i-2], ..., X[i-k]\}$
od

- Neighboring blocks share data:

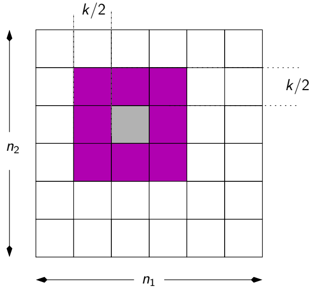


Shared Data: 2Dim

- Data parallel loop with shared input data:

for $i := 1$ to n_1 do
  for $i := 2$ to n_2 do
    $Y[i_1, i_2] := f(X[i_1, i_2], V[i_1, i_2]);$ $V[i_1, i_2] = \{X[i_1 - 1, i_2], X[i_1, i_2 - 1], ..., X[i_1 - k, i_2]\}$
  od

- symmetric window,



Optimal Granularity : 1Dim Data

Compare strategies for transferring shared data:

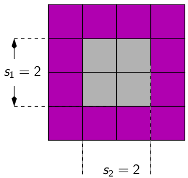
1. Replication: via *DMA transfers* from the off-chip memory to local memory.
2. Inter-processor communication: processors exchange data via the *network-on-chip* between the cores;
3. Local buffering: via *local copies* done by the processors.

Based on a parametric study, we derive optimal strategy and granularity for transferring shared data,

Optimal Granularity : 2Dim Data

- we consider Replication for transferring shared data,
- R: size of replicated data.

$R = 12$



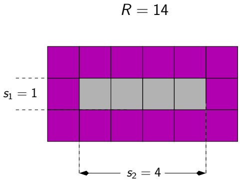
$(s_1, s_2) = (2, 2)$

Optimal Granularity : 2Dim Data

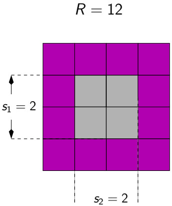
Influence of the block shape on the size of share data:

- Compare Transfer cost of a flat and a square block,

R: size of replicated data.



$(s_1, s_2) = (1, 4)$



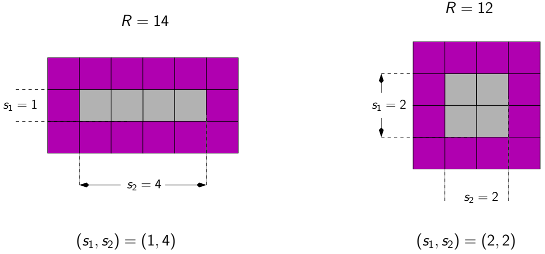
$(s_1, s_2) = (2, 2)$

Optimal Granularity : 2Dim Data

Influence of the block shape on the size of share data:

- Compare Transfer cost of a flat and a square block,

R: size of replicated data.



⊖ More Replicated data

Overhead

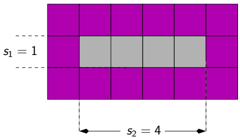
Optimal Granularity : 2Dim Data

Influence of the block shape on the size of share data:

- Compare Transfer cost of a flat and a square block,

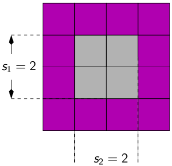
R: size of replicated data.

$R = 14$



$(s_1, s_2) = (1, 4)$

$R = 12$



$(s_1, s_2) = (2, 2)$

⊖ More Replicated data Overhead

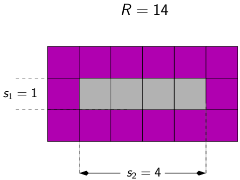
⊖ More transfer lines Overhead

Optimal Granularity : 2Dim Data

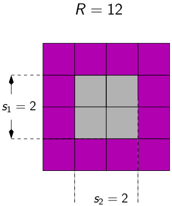
Influence of the block shape on the size of share data:

- Compare Transfer cost of a flat and a square block,

R: size of replicated data.



$(s_1, s_2) = (1, 4)$



$(s_1, s_2) = (2, 2)$

⊖ More Replicated data Overhead

⊖ More transfer lines Overhead

Optimal Granularity : 2Dim Data

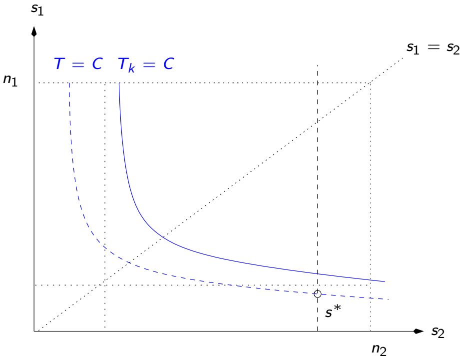
Problem Formulation

Find (s_1^*, s_2^*) such that,

$$\min \; T(s_1 + k, s_2 + k) \;\; \text{s.t.}$$

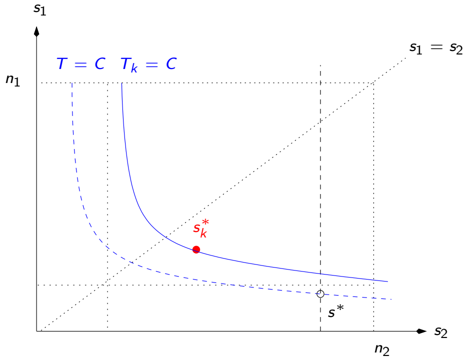
$$T(s_1 + k, s_2 + k) \leq C(s_1, s_2)$$
$$(s_1, s_2) \in [1..n_1] \times [1..n_2]$$
$$(s_1 + k) \times (s_2 + k) \leq M$$

Optimal Granularity : 2Dim Data



Optimal Granularity : 2Dim Data

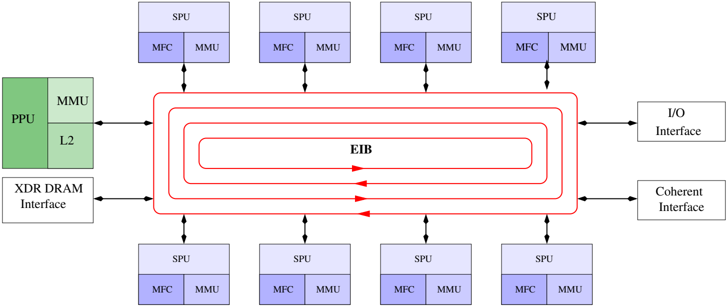
Optimal shape: between a square and a flat shape,



$$\begin{cases} s_1^* = \Delta + (c_1/\psi)(1/D) \\ s_2^* = \Delta + (l_1/\psi)(1 + D) \end{cases}$$

Outline

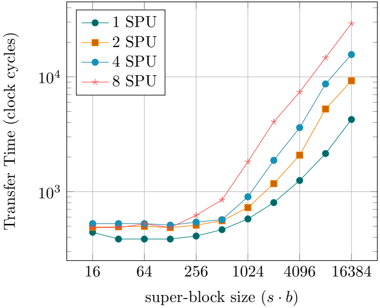
Overview of Cell B.E. Architecture



Platform Characteristics:

- 9-core heterogeneous multi-core architecture, with a Power Processor Element(PPE) and 8 Synergistic Processing Elements(SPE).
- Limited local store capacity per SPE: 256 Kbytes
- Explicitely managed memory system, using DMAs

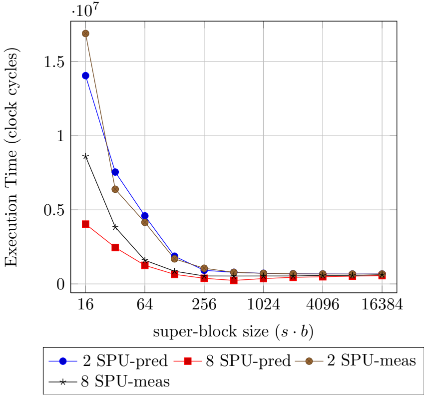
Measured DMA Latency



- Profiled hardware parameters:

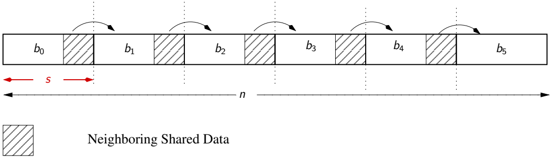
DMA issue time	I	$\backsimeq$ 400 clock cycles
Off-chip memory transfer cost/byte: 1 proc	$\alpha(1)$	0.22 clock cycles
Off-chip memory transfer cost/byte: p procs	$\alpha(p)$	$\backsimeq p \cdot \alpha(1)$
inter-processor comm transfer cost/byte for	β	0.13 clock cycles

Optimal Granularity: 1Dim Data, No Sharing

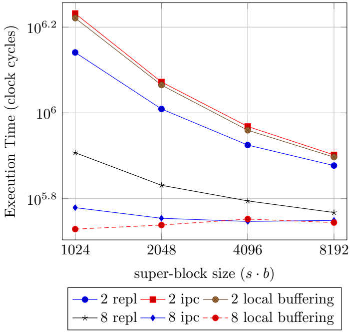


- predicted optimal granularities give good performance.

Optimal Granularity: 1Dim Data, Sharing

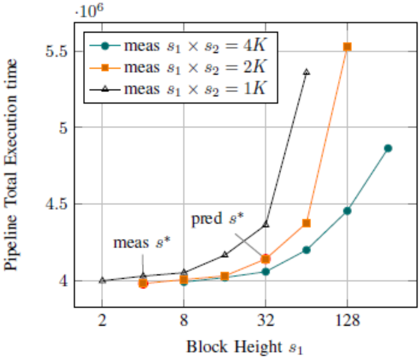


Comparing several strategies:



Optimal Granularity: 2Dim Data, Sharing

- We implement double buffering on a mean filtering algorithm,

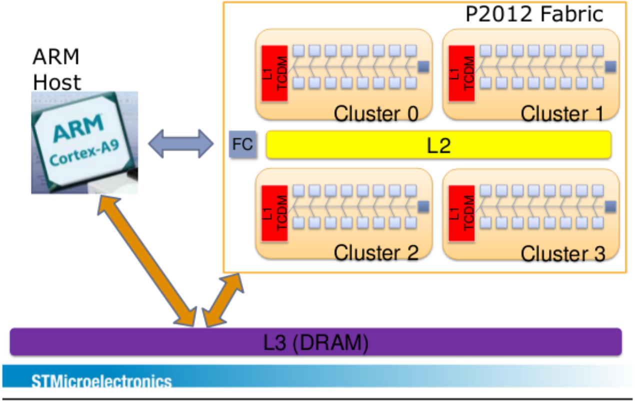


- predicted optimal granularities give good performance.

Outline

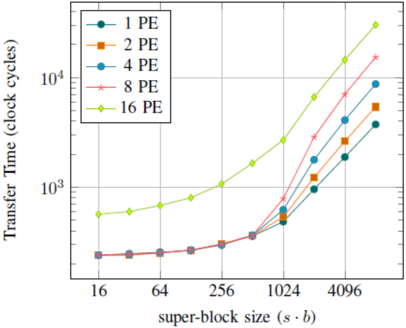
P2012 Memory Hierarchy

1. Intra cluster L1 memory (256 Kbytes),
2. Inter cluster L2 memory,
3. Off-chip L3 memory



DMA Latency

we measure the DMA latency on P2012,



DMA performance Model:

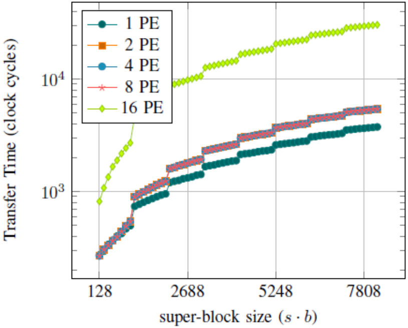
$$T(s,p) = I + \alpha(p)bs$$

$$I = 240 cycles$$

p	$\alpha(p)$
1	0.25
2	0.45
4	0.65
8	1.15
16	2.15

DMA Latency

we measure the DMA latency on P2012,



DMA performance Model:

$$T(s,p) = I + \alpha(p)bs$$

$I = 240 cycles$

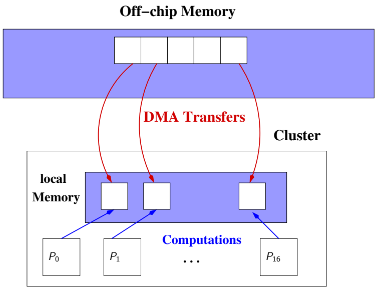
p	$\alpha(p)$
1	0.25
2	0.45
4	0.65
8	1.15
16	2.15

DMA Transfers in a Cluster

Shared Local Memory and shared DMA:
2 approaches for transferring data,

1. Liberal: Processors fetch data independently
2. Collaborative: Processors fetch data collectively

Liberal Approach: Processors fetch data independently



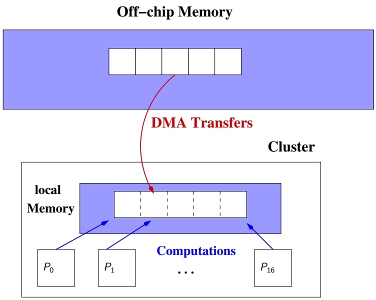
$$T(s,p) = I + \alpha(p)bs$$
$$C(s,p) = o + \omega s$$
$$s \leq M/p$$

Program 1 (Liberal): Kernel(data_type **global** *GBuffer, param1, param2 ,...)

```
{
data_type LBuffer [size];
...
async_work_item_copy(GBuffer, LBuffer, size, e);
...
wait_event(e);
}
```

Opencl Kernel:
work item = processor
async_work_item_copy: DMA fetch for each processor

Collab Approach: Processors fetch data Collectively



$$T(s,p) = I + \alpha(1)bs$$
$$C(s,p) = o(p) + (\omega/p)s$$
$$s \leq M$$

Program 2 (Collaborative): Kernel (data_type **global*** GBuffer, data_type **local** *LBuffer, param1, param2 ,...)

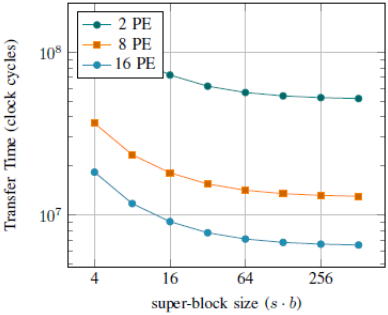
```
{
...
async_work_group_copy(GBuffer, LBuffer, size, e);
...
wait_group_event(e);
}
```

Opencl Kernel:
work group = cluster
async_work_group_copy: DMA fetch for the cluster

Liberal Approach: Processors fetch data independently

⊖ increase of number of processors reduces max buffer size,

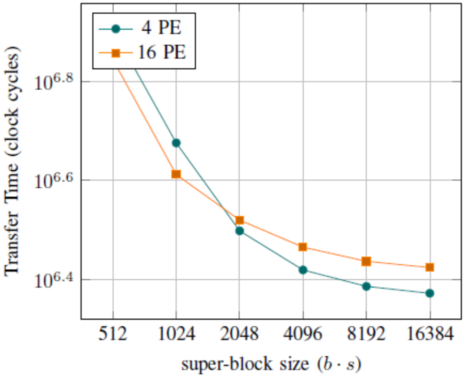
- double buffering implementation results:
  Optimal Granularity does not fit in the available memory space



Liberal Approach: Processors fetch data independently

⊖ synchronization overhead,

- double buffering implementation results:
  Performance degradation when increase number of processors



On-going Work:

- find the right balance between number of processors and Memory space budget,
- compare both liberal and collaborative approach,

Outline

Conclusion

- We presented a general methodology for automating decisions about Optimal granularity for data transfers,
- we capture the following facts,
  1. Block shape and size influence the transfer/computation Ratio,
  2. DMA performance (sensitivity to the block shapes, number of processors)
  3. tradeoff between Strided DMA overhead vs size of replicated data

Perspectives

1. Consider other applications patterns,
2. Capture variations (hardware and Software),
3. Generalize the approach to more than 2 memory levels,
4. Integrate this work in a complete compilation flow,
5. combine task and data parallelism,
6. ...

Thank You !!!